

CHANGING A SCHEDULER IN A VIRTUAL MACHINE MONITOR

BACKGROUND

[0001] A virtual machine (VM) architecture logically partitions a physical machine, such that the underlying hardware of the machine is time-shared and appears
5 as one or more independently operation virtual machines. A computer platform in a virtual machine environment may comprise a virtual machine monitor (VMM) that may create a plurality of virtual machines and runs on the computer platform to facilitate for other software the abstraction of one or more virtual machines.

10[0002] The virtual machine monitor may comprise a scheduler to allocate time slots for each virtual machine to run and prioritize or balance the resource usage among the virtual machines. Usually, a scheduler may implement a specific scheduling mechanism that may fit specific situations, such as Borrowed Virtual Time (BVT) algorithm, Round Robin algorithm, etc.

15 BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The invention described herein is illustrated by way of example and not by way of limitation in the accompanying figures. For simplicity and clarity of illustration, elements illustrated in the figures are not necessarily drawn to scale. For example, the dimensions of some elements may be exaggerated
20 relative to other elements for clarity. Further, where considered appropriate, reference labels have been repeated among the figures to indicate corresponding or analogous elements.

- [0004] Fig. 1 shows an embodiment of a computer platform having a virtual machine monitor to change a scheduler.
- [0005] Fig. 2 shows an embodiment of a scheduler manager in the virtual machine monitor of Fig. 1.
- 5[0006] Fig. 3 shows an embodiment of a method of changing a scheduler in the virtual machine monitor of Fig. 1.
- [0007] Fig. 4 shows an embodiment of a method of handling a scheduling request by the scheduler changed in Fig. 3.
- [0008] Fig. 5 shows an embodiment of a general computer platform having the
10 virtual machine monitor of Fig. 1.

DETAILED DESCRIPTION

- [0009] The following description describes techniques for changing a scheduler in a virtual machine monitor. In the following description, numerous specific details such as logic implementations, pseudo-code, means to specify
15 operands, resource partitioning/sharing/duplication implementations, types and interrelationships of system components, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the current invention. However, the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full
20 software instruction sequences have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation.

[0010] References in the specification to "one embodiment", "an embodiment", "an example embodiment", etc., indicate that the embodiment described may include a particular feature, structure, or characteristic, but every embodiment may not necessarily include the particular feature, structure, or characteristic.

5 Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to effect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly
10 described.

[0011] Embodiments of the invention may be implemented in hardware, firmware, software, or any combination thereof. Embodiments of the invention may also be implemented as instructions stored on a machine-readable medium, that may be read and executed by one or more processors. A machine-readable
15 medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computing device). For example, a machine-readable medium may include read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other forms of
20 propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.) and others.

[0012] An embodiment of a computer platform 10 having a virtual machine monitor to change a scheduler is shown in Fig. 1. The computer platform 10 may comprise an underlying hardware machine 11 having one or more processors
25 111, memory 112, console device 113, timer 114, and the like. The computer

platform 10 may further comprise a plurality of virtual machines and a virtual machine monitor. The plurality of virtual machines run their own operating systems and application software, such as a service virtual machine 13 running a service operating system 131 and a service application 132 and one or more guest virtual machines 14₁-14_N running their own guest operating systems 141₁-141_N and guest applications 142₁-142_N. The virtual machine monitor 12 may be responsible for processor(s)/memory virtualization/simulation, interrupt handling, virtual machine scheduling, etc. A non-exhaustive list of examples for the computer platform 10 may include mainframe computer, mini-computer, personal computer, portable computer, laptop computer and other devices for transceiving and processing data.

[0013] Various components in the computer platform 10 may invoke the virtual machine monitor 12 to perform a scheduling process, e.g., to determine a next owner of underlying switch devices for a next assigned time slot, or to determine a priority of a virtual machine, etc. Usually, the switch devices may be owned by a running virtual machine (e.g., a service virtual machine 13, or a guest virtual machine 14₁-14_N) in a time slot assigned by a scheduler in the virtual machine monitor and therefore have a focus on which virtual machine is currently active. Examples of the switch device may comprise the processor 111 and console device 113, wherein the console device 13 may comprise a frequently used I/O device, such as a keyboard, mouse, etc.

[0014] The service operating system 131 in the service virtual machine 13 may send a scheduling request for the scheduling process to the virtual machine monitor 12. For example, when the service operating system 131 monitors a running guest virtual machine 14₁-14_N, and finds a failure in the guest virtual

machine, e.g., a guest operating system fault, the service operating system 131 may send a scheduling request to the virtual machine monitor 12 to change the owner of the switch devices.

[0015] The guest operating system 141₁-141_N in the guest virtual machine 14₁-14_N may also send a scheduling request to the virtual machine monitor 12. For example, when the guest operating system 141₁-141_N is executing a device input/output operation and waiting for a response from the device, the guest operating system may send the scheduling request to yield the ownership of the switch devices so that other virtual machine may make a use of the switch devices.

[0016] The timer 114 in the underlying hardware, e.g., a programmable interval timer, may also send a scheduling request to the virtual machine monitor 12. For example, when the time slot assigned to the running virtual machine expires, the timer 114 may send a timer interrupt to the virtual machine monitor that may invoke the virtual machine monitor to perform the virtual machine scheduling in order to change the ownership of the switch devices.

[0017] The virtual machine monitor 12 may comprise a scheduler loader 120 and scheduler manager 121. The virtual machine monitor 12 may further comprise one or more schedulers; however, one of the one or more schedulers is active to handle the scheduling request. In the embodiment as depicted in Fig. 1, the virtual machine monitor 12 may comprise an old scheduler 122 and a new scheduler 123 that may respectively implement a specific scheduling mechanism such as Borrowed Virtual Time (BVT) algorithm, Round Robin algorithm, etc.

[0018] Many technologies may be applied as a working mode for the old scheduler 122 and the new scheduler 123. For example, the virtual machine monitor 12 may always hold the old scheduler 122 as a default scheduler. Before the new scheduler 123 is loaded in the virtual machine monitor 12, the old scheduler 122 may be active to handle the scheduling request. After the new scheduler 123 is loaded, the new scheduler may be active to handle the scheduling request in place of the old scheduler. However, if the new scheduler 123 is unloaded from the virtual machine monitor 12, the old scheduler 122 may be active again until the virtual machine monitor 12 is loaded with another new scheduler. For another example, in some circumstances, the scheduler manager 121 may switch back to the old scheduler 122 even though the new scheduler 123 exists in the virtual machine monitor 12. For still another example, the virtual machine monitor 12 may unload the old scheduler 122 before or after loading the new scheduler 123.

15[0019] The scheduler loader 120 may process a scheduler loading request from the service operating system 131 and load the new scheduler 123 in the virtual machine monitor 12 when the virtual machine monitor 12 or one or more of the virtual machines 13, 14₁-14_N is running. Software images of the new scheduler 123 may be available from various resources, such as a website, local disk, data center image server, etc. Example for an implementation of the scheduler loader 120 may comprises a hypercall handler that may process a hypercall for scheduler loading from the service operating system 131 and load the software images of the new scheduler 123 in the virtual machine monitor 12.

[0020] The scheduler manager 121 may be responsible for activating one of the old scheduler 122 and the new scheduler 123 to handle the scheduling request

when the virtual machine monitor 12 or one or more of the virtual machines 13, 14₁-14_N is running. The scheduler manager 121 may implement the scheduler activating in various ways. For example, the scheduler manager 121 may store a scheduler identifier to identify the scheduler that is active for the scheduling request. Before the new scheduler 123 is loaded in the virtual machine monitor 12, the scheduler manager 121 may store the old scheduler identifier in order to activate the old scheduler 122 to handle the scheduling request. After the new scheduler 123 is loaded, the scheduler manager 121 may replace the old scheduler identifier with the new scheduler identifier in order to activate the new scheduler to handle the scheduling request. However, when the new scheduler 123 is unloaded, the scheduler manager 121 may restore the old scheduler identifier to re-activate the old scheduler 122.

[0021] For another example, the scheduler manager may store a function pointer array pointing to a function array of the active scheduler, e.g., a function array of the old scheduler 122 or the new scheduler 123 which is active to handle the scheduling request. Fig. 2 depicts an embodiment for an implementation of the scheduler manager 121. As depicted, the scheduler manager 121 may comprise the function pointer array having a plurality of function pointers (e.g., pointers 0, 1, 2, 3, etc.). Each of the old scheduler 122 and new scheduler 123 may perform the scheduling process with a plurality of routine functions that adhere to a particular application programming interface (API), e.g., functions 0', 1', 2', etc. of the old scheduler 122 or functions 0'', 1'', 2'', etc. of the new scheduler 123. Each pointer in the function pointer array 121 may point to an active scheduler function, e.g., a function of the old scheduler 122 or the new scheduler 123. The scheduling requester, such as a virtual machine 13 or 14₁-

14_N, timer 114 or other devices that may trigger a scheduling process in the virtual machine monitor 12, may include a pointer to the function pointer array and may call the active scheduler function by dereferencing its pointer to the function pointer array and then calling the functions pointed by the array.

5[0022] In the embodiment depicted in Fig. 2, the scheduler manager 121 may activate the old scheduler 122 or the new scheduler 123 by updating the function pointer array to point to the active scheduler functions, or by updating the pointer of the scheduling requester to point to the function pointer array.

[0023] Referring back to Fig. 1, the scheduler manager 121 may be further
10 responsible for transporting information between the active scheduler (e.g., the old scheduler 122 or the new scheduler 123) and the scheduling requester. For example, in response to receiving the scheduling request from the scheduling requester, the scheduler manager 121 may dispatch the scheduling request to the active scheduler identified by the scheduler identifier or to the
15 active scheduler functions pointed by the function pointer array. The scheduler manager 121 may further send scheduling feedback information from the active scheduler to the scheduling requester. The scheduling feedback information may comprise a notification of the desired scheduling operation is performed correctly or not, a virtual machine priority information, and so on.

20[0024] However, other embodiments may implement other technologies for the structure of the computer platform 10. For example, the scheduler manager 121 may be omitted and the scheduling requester may issue a direct request to the virtual machine monitor requiring the scheduling process, wherein addresses associated to the active scheduler are dynamically patched into the

request during a scheduler model loading/unloading stage. With this means, the request may be sent to the active scheduler directly.

[0025] Fig. 3 depicts an embodiment of a method of changing a scheduler in the virtual machine monitor 12 as shown in Fig. 1. In block 301, a user or other suitable party may decide to change an old scheduler (e.g., the old scheduler 122) currently active to handle a scheduling request in the virtual machine monitor with a new scheduler (e.g., the scheduler 123) when the virtual machine monitor or one or more of the virtual machines is running. For example, a user may determine that the old scheduler is a poor fit for the current virtualization environment or that another scheduler is a better fit for the current virtualization environment.

[0026] In block 302, the decision made in block 301 may invoke an application running over a service operating system in a service virtual machine (e.g., service operating system 131) with specified parameters and the application may pass the parameters and other information to the service operating system through a virtual machine control request, that may trigger the service operating system to issue a scheduler loading request into the virtual machine monitor to load the new scheduler. The specific parameters may comprise information on which scheduler is to be loaded, where the scheduler image is; what kind of loading policy the virtual machine monitor may apply to load the new scheduler, etc.

[0027] In block 303, a scheduler loader (e.g., the loader 120) or other suitable component may handle the loading request and cease all of switch devices owned by a running virtual machine, wherein the switch devices may comprise processor(s) and console devices. Various methods may be applied to perform

the ceasing process. For processor ceasing, a cease sign may be input in all of virtual machine resume paths between all of virtual processor(s) in the virtual machine monitor and the running virtual machine. For console device ceasing, the virtual machine monitor may flush all of outstanding traffic by the help of console device models before the virtual machine monitor actually stop them. After the switch devices are ceased, the switch devices may reach a stable, consistent or predictable state so that the virtual machine monitor may retain their state for a next scheduling process.

[0028] In block 304, the scheduler loader or other suitable component may decide whether to unload the old scheduler from the virtual machine monitor before loading the new scheduler. In different circumstances, the scheduler loader may make different decision. For example, the scheduler loader may decide to reserve the old scheduler for a future use. However, if there is no free space for the new scheduler, the scheduler loader may decide to unload the old scheduler. In response to deciding to reserve the old scheduler, the scheduler loader or other suitable component may load the new scheduler in the virtual machine monitor with use of the parameters from the service operating system in block 306. However, in response to deciding to unload the old scheduler, the scheduler loader or other suitable component may unload the old scheduler from the virtual machine monitor in block 305 and then load the new scheduler in the virtual machine monitor in block 306. Other embodiments may implement other technologies for the scheduler unloading. For example, the scheduler loader may make a decision on whether to unload the old scheduler after the new scheduler is loaded.

[0029] In block 307, the new scheduler may be activated to handle the scheduling request in place of the old scheduler. For example, the scheduler manager 121 may activate the new scheduler by means of storing an identifier to identify the new scheduler or a function pointer array pointing to the new scheduler functions. For another example, the new scheduler may be activated by dynamically patching addresses associated with the new scheduler in the scheduler request so that a scheduling requester may directly call the new scheduler to perform the scheduling process.

[0030] Fig. 4 depicts an embodiment of a method of handling a scheduling request by the new scheduler changed in Fig. 3. In block 401, the virtual machine monitor may receive the scheduling request from the scheduling requester requiring a scheduling process. In block 402, the scheduling request may be transferred to the new scheduler, for example, through a scheduler manager (e.g., the scheduler manager 121) that may store an identifier to identify the new scheduler or a function pointer array pointing to a function array of the new scheduler.

[0031] In block 403, the new scheduler may handle the scheduling request, for example, determine a next owner of the switch devices or calculate a scheduling priority for a specific virtual machine. In block 404, the new scheduler may return scheduling feedback information to the scheduling requester. The scheduling feedback information may comprise a notification of the desired scheduling process is performed correctly or not, a virtual machine priority information, and so on.

[0032] Fig. 5 depicts an embodiment of a general computer platform having the virtual machine monitor as shown in Fig. 1. The computing platform may

comprise one or more processors 50, memory 51, chipset 52, I/O device 53, BIOS firmware 54 and the like. The one or more processors 50 are communicatively coupled to various components (e.g., the memory 51) via one or more buses such as a processor bus. The processors 50 may be
5 implemented as an integrated circuit (IC) with one or more processing cores that may execute codes under a suitable architecture, for example, including Intel® Xeon™, Intel® Pentium™, Intel® Itanium™ architectures, available from Intel Corporation of Santa Clara, California.

[0033] In an embodiment, the memory 51 may store codes to be executed by the
10 processor 50. A non-exhaustive list of examples for the memory 51 may comprise one or a combination of the following semiconductor devices, such as synchronous dynamic random access memory (SDRAM) devices, RAMBUS dynamic random access memory (RDRAM) devices, double data rate (DDR) memory devices, static random access memory (SRAM), flash
15 memory devices, and the like.

[0034] In an embodiment, the chipset 52 may provide one or more communicative path among the processor 50, memory 51 and various components, such as the I/O device 53 and BIOS firmware 54. The chipset 52 may comprise a memory controller hub 520, an input/output controller hub 521 and a firmware
20 hub 522.

[0035] In an embodiment, the memory controller hub 520 may provide a communication link to the processor bus that may connect with the processor 50 and to a suitable device such as the memory 51. The memory controller hub 520 may couple with the I/O controller hub 521 that may provide an
25 interface to the I/O devices 53 for the computing platform such as a keyboard

and a mouse. A non-exhaustive list of examples for the I/O devices 13 may comprise a keyboard, mouse, network card, a storage device, a camera, a blue-tooth, an antenna, and the like.

[0036] In an embodiment, the memory controller hub 520 may communicatively couple with a firmware hub 522 via the input/output controller hub 521. The 5 firmware hub 522 may couple with the BIOS firmware 54 that may store routines that the computing platform executes during system startup in order to initialize the processors 50, chipset 52, and other components of the computing platform. Moreover, the BIOS firmware 54 may comprise routines or 10 drivers that the computing device 1 may execute to communicate with one or more components of the compute platform.

[0037] The computer platform as depict in Fig. 5 may perform as the computer platform 10 as depicted in Fig. 1. The memory 51 may store software images as a virtual machine monitor including a scheduler loader, one or more 15 scheduler, and/or a scheduler manager. The memory 51 may further store service software including service operating system and service applications, and guest software including guest operating system and guest applications.

[0038] While certain features of the invention have been described with reference to example embodiments, the description is not intended to be construed in a 20 limiting sense. Various modifications of the example embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.